**4-2 Milestone Three: Algorithms and Data Structures Enhancement**

Alysha Pursley

Southern New Hampshire University

CS499: Computer Science Capstone

Professor Ramsey Kraya

July 27, 2025

**4-2 Milestone Three: Algorithms and Data Structures Enhancement**

**Artifact Overview**

This enhancement builds on the Python version of my Corner Grocer application, which was originally developed as a C++ program in CS 210. The program processes an input file containing purchased grocery items, counts the frequency of each item, and provides a menu-based interface for interacting with the results. In its previous version from Enhancement One, it allowed users to look up an item’s frequency, display a full list of items with their counts, and show a histogram of frequencies.

Enhancement Two expands the program’s capabilities by adding alphabetical sorting, frequency-based sorting, an item search feature, and two file export options for saving frequency data and histograms. The alphabetical sort lists items in A–Z order with their purchase counts, while the frequency sort arranges them from most to least purchased. The search function lets a user check whether a specific item exists and see its frequency. Two additional export features save the frequency list and histogram to files so that results can be archived or shared.

To maintain modularity and organization, all sorting and searching functions were placed into a separate module (grocery\_utils.py) while the main file remains responsible for menu handling and frequency processing. This separation of concerns ensures the program stays clean, scalable, and easy to maintain while making new features easier to test and debug.

***Justification for Inclusion***

This artifact is an excellent example for demonstrating algorithms and data structures because it relies on both for its core functionality. The program uses Python’s defaultdict(int) for efficient frequency counting and constant-time lookups. Alphabetical sorting uses the built-in sorted() function applied to dictionary keys, while frequency sorting uses a custom key to order dictionary items by value in descending order. The search function normalizes user input to lowercase before performing the lookup, ensuring accurate results regardless of case differences.

By placing all new sorting and searching logic in grocery\_utils.py, the design preserves a clean separation between business logic and the user interface. This modular structure not only makes the code easier to maintain and extend but also supports collaborative development. Importantly, all original functionality from Enhancement One was preserved without any regressions, ensuring backward compatibility while delivering new, valuable features.

***Skills and Course Outcomes***

1. **Course Outcome 1:**

*“Employ strategies for building collaborative environments that enable diverse audiences to support organizational decision-making in the field of computer science.”*

Some of the skills I’ve demonstrated in this enhancement that meet this outcome are:

* Modular Design
* Code Documentation
* Modular Architecture
* Code Readability
* Separation of Concerns

**How these skills support this outcome:**

* **Modular Design –** By placing sorting and searching functions into a separate grocery\_utils.py file, I made the codebase easier for other developers to understand and extend, promoting effective collaboration.
* **Code Documentation –** *D*etailed, structured comments clearly explain each function’s purpose and logic, ensuring that anyone reviewing the code can quickly understand its functionality.
* **Modular Architecture –** Separating the utility module from the main program supports a team-based workflow, allowing multiple developers to work on different parts of the program without conflict.
* **Code Readability –** Using clear naming conventions, logical organization, and consistent formatting makes it easier for others to follow the program’s structure.
* **Separation of Concerns –** Keeping business logic separate from menu and input handling ensures that changes can be made to one area without breaking unrelated functionality, simplifying collaborative work.

1. **Course Outcome 2:**

*“Design, develop, and deliver professional-quality oral, written, and visual communications that are coherent, technically sound, and appropriately adapted to specific audiences and contexts.”*

Some of the skills I’ve demonstrated in this enhancement that meet this outcome are:

* Menu Design
* User Experience Improvements
* Expanded Menu System
* Code Documentation
* Code Readability

**How these skills support this outcome:**

* **Menu Design –** The menu was expanded to nine options while remaining clear and logically organized, ensuring that users can quickly understand how to access each feature.
* **User Experience Improvements –** Clear prompts, error messages, and outputs make the program easy for any user to work, even without prior technical experience.
* **Expanded Menu System –** Adding new features without making the interface overwhelming shows careful attention to user communication.
* **Code Documentation –** Well-written comments act as a form of technical communication, making the intent and logic of the program easy to follow.
* **Code Readability –** Maintaining consistent formatting and descriptive variable/function names makes the code understandable to both technical reviewers and new developers.

1. **Course Outcome 3:**

*“Design and evaluate computing solutions that solve a given problem using algorithmic principles and computer science practices and standards appropriate to its solution while managing the trade-offs involved in design choices.”*

Some of the skills I’ve demonstrated in this enhancement that meet this outcome are:

* Data Sorting
* Alphabetical Sorting Logic
* Frequency-Based Sorting
* Separation of Concerns
* Function Abstraction
* Code Reusability

**How these skills support this outcome:**

* **Data Sorting –** Implementing both alphabetical and frequency-based sorting demonstrates the ability to design different algorithmic approaches for viewing the same dataset.
* **Alphabetical Sorting Logic –** Applying sorted() to dictionary keys reflects an understanding of efficient sorting methods.
* **Frequency-Based Sorting –** Using a custom sort key to order by frequency highlights control over sorting algorithms and data structures.
* **Separation of Concerns –** Keeping the sorting and searching logic separate from menu flow makes the codebase more maintainable and scalable.
* **Function Abstraction –** Encapsulating each sorting and searching feature in its own function allows for better control, testing, and modification.
* **Code Reusability –** Designing the sorting and searching logic so it can be reused across the program demonstrates a scalable problem-solving approach.

1. **Course Outcome 4:**

*“Demonstrate an ability to use well-founded and innovative techniques, skills, and tools in computing practices for the purpose of implementing computer solutions that deliver value and accomplish industry-specific goals.”*

Some of the skills I’ve demonstrated in this enhancement that meet this outcome are:

* Programming Proficiency
* Search Functionality
* Function Abstraction
* Code Reusability
* Preserved Backward Compatibility

**How these skills support this outcome:**

* **Programming Proficiency –** Designing and implementing new features using Python’s built-in tools ensures that the application delivers real-world value efficiently.
* **Search Functionality –** Adding the ability to find specific items quickly makes the application more useful for inventory tracking and analysis.
* **Function Abstraction –** Keeping each new feature in its own function improves maintainability and allows for easier enhancements in the future.
* **Code Reusability –** Writing generic sorting and searching functions means they can be adapted to different uses without rewriting logic.
* **Preserved Backward Compatibility –** Maintaining all original features while adding new ones ensures that existing workflows remain intact for users.

1. **Course Outcome 5:**

*“Develop a security mindset that anticipates adversarial exploits in software architecture and designs to expose potential vulnerabilities, mitigate design flaws, and ensure privacy and enhanced security of data and resources.”*

Some of the skills I’ve demonstrated in this enhancement that meet this outcome are:

* Input Validation
* Input Handling Enhancements
* User Experience Improvements

**How these skills support this outcome:**

* **Input Validation –** Menu selections are checked for valid ranges, and item names are restricted to letters and spaces, preventing invalid data from being processed.
* **Input Handling Enhancements –** Validation logic was updated to handle the expanded menu, catching errors before they can cause program crashes.
* **User Experience Improvements –** Clear error messages guide the user to correct mistakes without compromising program stability or security.

***Reflection on the Enhancement Process***

Completing this enhancement reinforced how valuable modular architecture is for maintainability and scalability. Moving sorting and searching into a separate utility module kept the main program focused on menu logic and data processing, making the code easier to read and update. It also made testing simpler since I could run and verify each helper function independently.

Another takeaway was the importance of balancing new features with interface clarity. Expanding the menu from four to nine options could have made the interface cluttered, but careful grouping and consistent formatting kept it user-friendly. Input validation remained a top priority, ensuring that no matter how the program grows, it continues to handle user input safely and predictably. Overall, this enhancement improved functionality, usability, and maintainability without sacrificing the stability of the original design.